ANALYSIS

44(202), October 1, 2015

ISSN 2278–5469 EISSN 2278–5450

Discovery

Analysis and proposition of fault-tolerance model for real time software systems

Ankur Ghartaan[1], Tirthankar Gayen[2]

1.Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi, India; Email:- ankur.ghartaan@gmail.com
2.Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi, India; Email:- tgayen77@gmail.com

Article History
Received: 02 August 2015
Accepted: 15 September 2015
Published: 1 October 2015

Citation
Ankur Ghartaan, Tirthankar Gayen. Analysis and proposition of fault-tolerance model for real time software systems. *Discovery*, 2015, 44(202), 62-67

ABSTRACT

Real time systems are those systems which must guarantee to response correctly within strict time constraint or within deadline. Failures can arise from both functional errors as well as timing bugs. Hence, it is necessary to provide temporal correctness of programs used in real time applications in addition to providing functional correctness. Although, there are several researches concerned with achieving fault tolerance in the presence of various functional and operational errors but many of them did not address the problem concerned with the timing bugs which is an important issue in real time systems. As for real time systems, many times it becomes a necessity for a given service to be delivered within the specified time deadline. Therefore, this paper analyses the shortcoming of the existing approaches with respect to real time systems and proposes a versatile, cost effective approach in the presence of timing bugs for providing fault tolerance to real time software applications.

Index Terms – timing bugs, real time, fault tolerance, deadline

1. INTRODUCTION

Reliability has always been an important quality attributes of software systems specially for the mission and safety critical software systems because in such systems severity of consequences resulting from failures is very high. Software is becoming more and more

complex with the passage of time to cope with the emerging application requirements. More complex the software is, more difficult it is to assure high reliability. Complex software has a large number of states (unlike the hardware), so it is not practically possible to completely test the software. Irrespective, of the amount of testing one can do, it can never be assured that the final software product is fault free. In order to achieve failure free operation of software, one develops some mechanism to handle the faults remaining in the system after the development. A real-time system is one that must produce a correct result within a specified time deadline. According to Hermann Kopetz, "*A real time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced*" [1]. Real time systems are time critical and their correctness depends on both their timeliness and the correctness of output. Young, in 1982 defined the real time system as, "*Any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period.*" Real-time systems are classified as hard real time systems and soft real time systems. In hard real time systems if deadline is missed consequences can be catastrophic but in soft real time systems consequences of missing the deadline are relatively tolerable. Some examples of real time systems are: aircraft control, oven temperature controller and over-temperature monitor in nuclear power station etc. In real time systems severity of consequences resulting from failures may be high. Sometimes failure may endanger the human lives. Today, a good amount of research is going on the techniques for development of safe and better quality software. Since real-time systems need to be highly reliable and as it is not always possible to ensure the development of highly reliable system therefore they are prime candidates for the inclusion of fault tolerance techniques [2].

2. RELATED WORK

Many techniques are currently in use to achieve fault tolerance in software like n-version programming, recovery blocks (RcB), n-self checking programming, n-copy programming, retry blocks (RtB), adaptive n-version systems, rejuvenation, fuzzy voting, etc, yet these techniques may not be very suitable for real time systems. Software fault tolerance techniques are mostly based on traditional hardware fault tolerance. The goal of software fault tolerance techniques is to allow the system to function properly in the presence of software faults remaining in the system after completing the development of software. Unlike hardware just redundancy is not enough to deal with software faults; some form of diversity is also required along with redundancy [4].

N-Version Programming (NVP) is one of the software fault tolerance techniques based on design diversity. Elmendorf in 1972 suggested the concept of NVP and later in 1977–1978 Avizienis and Chen developed it [5]. To achieve fault tolerance in NVP technique a decision mechanism (DM) and a forward recovery mechanism is used. At least two variants of a program which are independently designed and functionally equivalent are developed from the common specifications. All these variants called versions are run in parallel and results produced by each version is passed to the DM. Decision mechanism selects the best result after examining all the results. Now a day's various alternative decision mechanisms are available to be used with NVP. In 1974 Horning, et al. introduced Recovery Blocks (RcB) [15]. Its earlier implementations were made by Randell (1975) and Hecht (1981) [16]. The basic RcB scheme comprises of an executive, an acceptance test, and primary and alternate try blocks (variants). Based on the acceptance tests (AT) result RcB selects a variant result and pass it to the output during program execution. By using the primary alternate (or try block) the RcB technique will initially try to ensure the AT (e.g., a test is passed based on the acceptability of a result of an alternate). If the primary algorithm's result did not pass the AT, then n-1 alternatives will be tried (attempted) until an alternate's results pass the AT. An error occurs when any of the alternates cannot pass the AT [2]. Laprie, et al developed N-Self-Checking Programming (NSCP) which is a design diverse technique. A self-checking program uses program redundancy to check its own behavior during execution .The NSCP hardware architecture comprises of four components grouped in two pairs in hot standby redundancy, where one software variant is supported by each hardware component. NSCP software contains a comparison algorithm and two variants or an AT and one variant on each hardware pair [3].

Data diversity as a complementary software fault tolerance strategy to design diversity was proposed by Ammann and Knight in 1987 [6]. Data diversity based techniques implements diversity at the input data. To achieve data diversity every data diversity based technique uses a data re-expression algorithm (DRA). DRA produces diverse input data sets that are logically equivalent. The performance of data diversity is highly dependent on the data re-expression algorithm. The problem associated with data diverse approaches is that all applications can not employ data diversity because it is not possible to find an effective Data re-expression algorithm for every application. Ammann and Knight developed Retry Blocks (RtB) which is one of the data diverse software fault tolerance techniques [6]. AT and backward recovery to achieve fault tolerance is used by RtB technique. A watchdog timer is used to trigger the execution of secondary algorithm if the actual algorithm fails to generate a desired result within a specified time period. Later Ammann and Knight also developed N-Copy Programming (NCP) as a data diversity technique [6]. The NCP technique uses a forward recovery and a decision mechanism to accomplish fault tolerance. At least two copies of a program and at least one data re-

expression algorithm is used by NCP. System inputs are passed to a DRA in order to generate logically equivalent input data sets. Using input the re-expressed data the copies execute in parallel. A decision mechanism works to examine the results of the copy executions and if there exists a best result, it selects it. NCP is considered as a data diverse complement of NVP [6].

The new software fault tolerance techniques are fuzzy voting, Byzantine fault tolerance, adaptive N-version systems and graph reduction. Kanoun, K., et al. considered modified classical n-version systems by incorporating in each version an individual weight factor. It considers an adaptive approach to model and manage different quality levels of the versions. This weight factor is then incorporated in the voting procedure, i.e. for the deviation behavior of the individual version the voting is based on a weighted counting of the number of monitored events [3]. In order to deal with transient software failures caused by software aging software rejuvenation is a novel approach, which can be considered as a proactive and preventive solution to counteract the aging phenomenon. Rejuvenation Involves occasional stoppage of the running software, restarting it and cleaning of the internal state. Cleaning of the system's internal state may involve flushing kernel tables of operating system, garbage collection and to reinitialize the internal data structures, etc. Hardware reboot is well known example of rejuvenation. Trivedi, T. S.A considered fault-tolerant software system as a redundant structure of two-versions and random schedule of rejuvenation. A quantitative evaluation of the steady-state system availability was done using Markovian analysis [4].

Because of the difficulty in justification of the extra cost of up-front development the traditional fault tolerance techniques are usually avoided in the development of mission critical systems. In the year 1996 Robert J Kreutzfeld, et. al. [7] presented a methodology called Data Fusion Integrity Process (DFIP) as an alternative for the traditional software fault tolerance techniques in order to deal with high sunk cost of development. This is a simple and effective technique for the development of fault tolerant mission critical systems. A DFIP implementation includes one recover, one detection, and the report method. The recover method's implementation realizes biggest cost saving in DFIP. The modular approach of DFIP makes it easier to implement at any time. Many software fault tolerance techniques are available but the relative effectiveness of different techniques remains unclear. S. Garnaik, et al., [9] in 2013 presented an approach for reliability enhancement of software programs by minimizing the overflow errors. They handled the problem of overflow because of large size integer input. The presented approach says that instead of storing the large size integer input in primary into data type; store it in the linked list data structure. Every node of this linked list contains two digits and pointer to next node. Authors present technique for addition and multiplication operation of large integer input using linked list. This approach provides tolerance to the faults occurring due to the use of large size data to the extent that as long as the free memory is available for allocation there won't be any failure due to use of large size data.

2. ANALYSIS FOR REAL TIME SYSTEMS

N-Version programming both in the presence and absence of data diversity is not applicable for situations in which distinct multiple solutions exist. Majority voting for N-Version programming may result in incorrect decision when majority of outputs are incorrect. Design diversity may not be very effective if similar kind of faults get detected in various versions. Implementation of effective design diversity may incur high cost due to the development of many versions. Hence it is not a cost effective approach for real time systems. Though implementation of data diversity based N-copy programming using data re-expression algorithm is a cost effective approach, yet it is not very useful as it is difficult to obtain an effective data re-expression algorithm for various real time system. The effectiveness of various fault tolerance techniques varies with variations in systems. Therefore there is no universal fault tolerance technique which is effective for all real time systems. There is no methodology to assess the relative effectiveness of different fault tolerance techniques. Recovery block technique is not suitable for real time systems due to its serial nature of execution.

From the survey, it has been found that there are several works concerned with achieving fault tolerance in the presence of various functional and operational errors but many of them did not address the problem concerned with timing bugs which is an important issue in real time systems. As for real time systems, many times it becomes a necessity for a given service to be delivered within the specified time deadline. Therefore the main objective is to develop a versatile cost effective approach in the presence of timing bugs for providing fault tolerance to real time software applications.

3. PROPOSED APPROACH

The proposed approach is based on the flow diagram provided in Fig.1. Here, the entire software system is divided into subsystems. Risk assessment is done on various tasks for each subsystem to identify those tasks which are more critical. From the task dependency analysis deadline is obtained for each subsystem. When a subsystem is executed, both the original program and trained neural network (TNN) module is executed simultaneously for the critical tasks of that subsystem. If there is any incomplete/missing data then the prediction module is used to provide the missing data. There is a constant polling to check whether the output is available from the original program. If the output is available from the original block before the time (deadline – k) (where k is the

reasonable amount of time required for completing the operations concerned with delivering the appropriate output for subsequent processing and deadline corresponds to the deadline time for the task completion) then it is forwarded further for subsequent processing else if the output is available from the TNN block then it is forwarded for subsequent processing. Otherwise, the subsystem moves to the safe mode and sends appropriate signal for subsequent processing. The delay in producing the output from the original program may depend on lot of factors like task dependency, resource sharing, unavailability of the required data etc. The detailed steps concerned with this approach are specified in algorithm fault_tol (system).

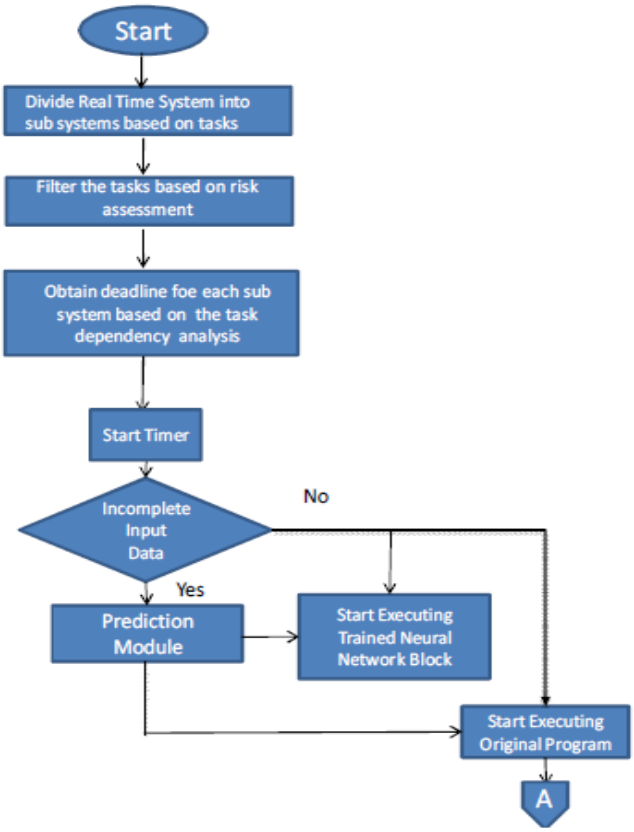


Figure 1 Flow diagram of the proposed approach

The proposed algorithm is specified in the following steps.

Algorithm *fault_tol(system)*

{

Step 1: Divide the real time system into subsystems based on tasks.

Step 2: Filter the task based on risk assessment.

Step 3: Obtain deadline for each sub system based on the task dependency analysis.

Step 4: For each sub system repeat step 5 to step 10.

Step 5: Start the timer.

Step 6: If the input data set is complete then pass input to TNN block and original program. Else pass incomplete input to prediction module and then pass predicted complete input data to TNN block and original program.

Step 7: Start execution of TNN block and original program simultaneously.

Step 8: If timer < (Deadline – k)

Step 8.1: If the output of the original program is available then forward it for subsequent processing else go to step 8.1.

Step 9: If timer > = (Deadline – k)

Step 9.1: If the output from the trained neural network block is available then forward it for subsequent processing else goto step 10.

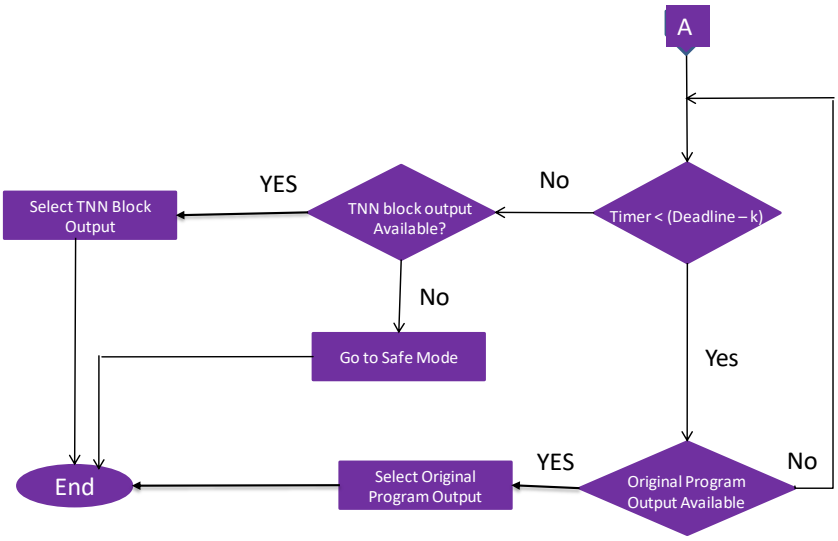
Step 10: Move the subsystem to the safe mode and send appropriate signal for subsequent processing

}

Where

k is the reasonable amount of time required for completing the operations concerned with delivering the appropriate output for subsequent processing

deadline corresponds to the deadline time for the task completion.



4. DISCUSSION

Since, the delay in producing the output from the original program may depend on lot of factors like task dependency, resource sharing, unavailability of the required data etc. The TNN block is expected to deliver the output within the specified deadline. Under, the extreme situation when the TNN block is unable to deliver its output within the specified deadline provision has been made to make the subsystem work in safe mode so that appropriate precautions can be taken to avoid any disaster or the loss can be minimized. Currently, an approach has been developed to achieve fault tolerance in the presence of timing bugs for real time software applications. Work is in progress for applying this approach to various existing real time applications and obtaining the results. Some of the results obtained were found to be better than other existing approaches especially in the presence of timing bugs.

5. CONCLUSION

A real-time system should process information and produce a response within a specified time. Real time systems are time critical and their correctness depends on both the correctness of output and their timeliness. In real time systems failure may cause very severe consequences. Since it is not always possible to ensure the development of highly reliable real time systems hence they are prime candidates for the inclusion of fault tolerance techniques. In this paper, the shortcoming of the existing approaches with respect to real time systems has been analyzed to propose a versatile and cost effective approach in the presence of timing bugs for providing fault tolerance to real time software applications.

ACKNOWLEDGMENT

The authors would like to thank the Dean, and all the staffs of SC & SS, JNU for providing direct or indirect support for this work.

REFERENCE

1. Kopetz, H., "Real-Time Systems: Design Principles for Distributed Embedded Applications", Wiley, 2nd edition, 2006
2. Anderson T. and Knight J.C., "A Framework for Software Fault Tolerance in Real-Time Systems," IEEE Transactions on Software Engineering, vol. se-9, no. 3, May 1983

3. Avizienis, A. and Kelly, J.P.J., "Fault Tolerance by Design Diversity: Concepts and Experiments," IEEE Computer, Vol. 17, No. 8, 1984, pp. 67–80.
4. Zaipeng, X., Sun, H. and Saluja, K.,"A Study of Software Fault Tolerance Techniques," 1415 Engineering Drive, Madison WI 53706, USA 1998s
5. Chen, L. and Avizienis, A., "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," Proceedings of FTCS- 8, Toulouse, France, pp. 3–9, 1978
6. Ammann, P. E. and Knight, J.C., "Data Diversity: An Approach to Software Fault Tolerance," IEEE Transactions on Computers, Vol. 37, No. 4, pp. 418 – 416, 1988.
7. Robert, J. K. and Neese, R. E., "A Methodology for Cost Effective Software Fault Tolerance for Mission-Critical Systems," 15th AIAA/IEEE Digital Avionics Systems Conference, Atlanta, GA, pp. 19-24, 1996.
8. Guillermo, G., May, J. and Julio, C. G., "Assessment of Data Diversity Methods for Software Fault Tolerance Based on Mutation Analysis," Mutation 2006- ISSRE Workshops, IEEE, 2006.
9. Garnaik, S., Gayen, T., Mishra, S., "Reliability Enhancement of Software by Minimizing the Overflow Errors", International Journal of Systems Assurance Engineering and Management, Springer, Vol. 5, No. 4, pp. 724-730, 2014.
10. Johnson B.W. ,"Fault-Tolerant Microprocessor-Based Systems", IEEE Micro, vol. 4, no. 6, pp. 6-21, 1984.
11. Kopetz H., "Design Principles for Distributed Embedded Applications", Springer, 2011.
12. Ng., Y.W., Avizienis A., "A reliability model for gracefully degrading and repairable fault-tolerant systems", Proc. International Symposium on Fault-Tolerant Computing, pp. 22 -28 , 1977
13. Biswas S., Mall R., "Task Dependency Analysis for Regression Test Selection of Embedded Programs", IEEE Embedded Systems Letters, 2011.
14. ANSI/IEEE, "Standard Glossary of Software Engineering Terminology", STD-729-1991, ANSI/IEEE, 1991.
15. Horning, J.J., Lauer, H.C., Melliar Smith, P.M., Randell, B., "A program structure for error detection and recovery," in Oper. Syst., Proc. Int. Symp (Lecture Notes in Comput. Sci.), vol. 16, E. Gelenbe and C. Kaiser, Eds. Berlin: Springer-Verlag, pp. 171-187, 1974.
16. Hecht, H., "Fault tolerant software for real-time applications," Comput. Surveys, vol. 8, pp. 391-407, Dec. 1976.